period of time expires without any activity from the Mobile End System **104**, the Mobility Management Server **102** may terminate a session Also, an administrator may want to limit the overall time a particular connection may be established for, or when to deny access base on time of day Again these policy timers may, in one example implementation, be invoked only on the Mobility Management Server **102** side

In one example implementation, the software providing the Internet Mobility Protocol is compiled and executable under Windows NT, 9x, and CE environments with no platform specific modification To accomplish this, Internet Mobility Protocol employs the services of a network abstraction layer (NAL) to send and receive Internet Mobility Protocol frames Other standard utility functions such as memory management, queue and list management, event logging, alert system, power management, security, etc are also used A few runtime parameters are modified depending on whether the engine is part of an Mobile End System **104** or Mobility Management Server **102** system Some examples of this are:

- Certain timeouts are only invoked on the Mobility Management Server **102**
- Direction of frames are indicated within each frame header for echo detection
- Inbound connections may be denied if Mobile End System **104** is so configured
- Alerts only signaled on Mobility Management Server **102**
- Power management enabled on Mobile End System **104** but is not necessary on the Mobility Management Server **102**

The Internet Mobility Protocol interface may have only a small number of "C" callable platform independent published API functions, and requires one O/S specific function to schedule its work (other then the aforementioned standard utility functions) Communications with local clients is achieved through the use of defined work objects (work requests) Efficient notification of the completion of each work element is accomplished by signaling the requesting entity through the optional completion callback routine specified as part of the work object

The Internet Mobility Protocol engine itself is queue based Work elements passed from local clients are placed on a global work queue in FIFO order This is accomplished by local clients calling a published Internet Mobility protocol function such as "ProtocolRequestwork( )". A scheduling function inside of Internet Mobility Protocol then removes the work and dispatches it to the appropriate function Combining the queuing and scheduling mechanisms conceal the differences between operating system architectures—allowing the protocol engine to be run under a threaded based scheme (e g., Windows NT) or in a synchronous fashion (e g , Microsoft Windows 9x & Windows CE) A priority scheme can be overlaid on top of its queuing, thus enabling a guaranteed quality of service to be provided (if the underlying network supports it)

From the network perspective, the Internet Mobility Protocol uses scatter-gather techniques to reduce copying or movement of data Each transmission is sent to the NAL as a list of fragments, and is coalesced by the network layer transport If the transport protocol itself supports scatter-gather, the fragment list is passed through the transport and assembled by the media access layer driver or hardware Furthermore, this technique is extensible in that it allows the insertion or deletion of any protocol wrapper at any level of the protocol stack Reception of a frame is signaled by the NAL layer by calling back Internet Mobility Protocol at a specified entry point that is designated during the NAL registration process

Internet Mobility Protocol Engine Entry Points

Internet Mobility Protocol in the example embodiment exposes four common entry points that control its startup and shutdown behavior These procedures are:

1 Internet Mobility ProtocolCreate( )
2 Internet Mobility ProtocolRun( )
3 Internet Mobility ProtocolHalt( )
4 Internet Mobility ProtocolUnload( )

Internet Mobility ProtocolCreate( )

The Internet Mobility ProtocolCreate( ) function is called by the boot subsystem to initialize the Internet Mobility Protocol During this first phase, all resource necessary to start processing work must be acquired and initialized At the completion of this phase, the engine must be in a state ready to accept work from other layers of the system At this point, Internet Mobility Protocol initializes a global configuration table To do this, it employs the services of the Configuration Manager **228** to populate the table

Next it registers its suspend and resume notification functions with the APM handler In one example, these functions are only invoked on the Mobile End System **104** side—but in another implementation it might be desirable to allow Mobility Management Server **102** to suspend during operations Other working storage is then allocated from the memory pool, such as the global work queue, and the global NAL portal list

To limit the maximum amount of runtime memory required as well as insuring Internet Mobility Protocol handles are unique, Internet Mobility Protocol utilizes a 2-tier array scheme for generating handles The globalConnectionArray table is sized based on the maximum number of simultaneous connection the system is configured for, and allocated at this time. Once all global storage is allocated and initialized, the global Internet Mobility Protocol state is change to _STATE_INITIALIZE_

Internet Mobility ProtocolRun( )

The Internet Mobility ProtocolRun( ) function is called after all subsystems have been initialized, and to alert the Internet Mobility Protocol subsystem that it is okay to start processing any queued work This is the normal state that the Internet Mobility Protocol engine is during general operations A few second pass initialization steps are taken at this point before placing the engine into an operational state

Internet Mobility Protocol allows for network communications to occur over any arbitrary interface(s) During the initialization step, the storage for the interface between Internet Mobility Protocol and NAL was allocated Internet Mobility Protocol now walks through the global portal list to start all listeners at the NAL In one example, this is comprised of a two step process:

- Internet Mobility Protocol requests the NAL layer to bind and open the portal based on configuration supplied during initialization time; and
- Internet Mobility Protocol then notifies the NAL layer that it is ready to start processing received frames by registering the Internet Mobility ProtocolRCVFROMCB call back.

A local persistent identifier (PID) is then initialized

The global Internet Mobility Protocol state is change to _STATE_RUN_

Internet Mobility ProtocolHalt

The Internet Mobility ProtocolHalt( ) function is called to alert the engine that the system is shutting down. All resources acquired during its operation are to be release

prior to returning from this function. All Internet Mobility Protocol sessions are abnormally terminated with the reason code set to administrative. No further work is accepted from or posted to other layers once the engine has entered into _STATE_HALTED_state.

Internet Mobility ProtocolUnload( )

The Internet Mobility ProtocolUnload( ) function is the second phase of the shutdown process. This is a last chance for engine to release any allocated system resources still being held before returning. Once the engine has returned from this function, no further work will be executed as the system itself is terminating.

Internet Mobility Protocol Handles

In at least some examples, using just the address of the memory (which contains the Internet Mobility Protocol state information) as the token to describe an Internet Mobility Protocol connection may be insufficient. This is mainly due to possibility of one connection terminating and a new one starting in a short period of time. The probability that the memory allocator will reassign the same address for different connections is high—and this value would then denote both the old connection and a new connection. If the original peer did not hear the termination of the session (i.e. it was off, suspended, out of range, etc.), it could possibly send a frame on the old session to the new connection. This happens in TCP and will cause a reset to be generated to the new session if the peer's IP addresses are the same. To avoid this scenario, Internet Mobility Protocol uses manufactured handle. The handles are made up of indexes into two arrays and a nonce for uniqueness. The tables are laid out as follows:

Table 1: an array of pointers to an array of connection object.

Table 2: an array of connection objects that contains the real pointers to the Internet Mobility Protocol control blocks.

This technique minimizes the amount of memory being allocated at initialization time. Table 1 is sized and allocated at startup. On the Mobile End System **104** side this allows allocation of a small amount of memory (the memory allocation required for this Table 1 on the Mobility Management Server **102** side is somewhat larger since the server can have many connections).

Table 1 is then populated on demand. When a connection request is issued, Internet Mobility Protocol searches through Table 1 to find a valid pointer to Table 2. If no entries are found, then Internet Mobility Protocol will allocate a new Table 2 with a maximum of 256 connection objects—and then stores the pointer to Table 2 into the appropriate slot in Table 1. The protocol engine then initializes Table 2, allocates a connection object from the newly created table, and returns the manufactured handle. If another session is requested, Internet Mobility Protocol will search Table 1 once again, find the valid pointer to Table 2, and allocate the next connection object for the session. This goes on until one of two situations exist:

- If all the connection objects are exhausted in Table 2, a new Table 2 will be allocated, initialized, and a pointer to it will be placed in the next available slot in Table 1; and
- If all connection objects have been released for a specific Table 2 instance and all elements are unused for a specified period of time, the storage for that instance of Table 2 is released back to the memory pool and the associated pointer in Table 1 is zeroed to indicate that that entry is now available for use when the next connection request is started (if and only if no other connection object are available in other instances of Table 2).

Two global counters are maintained to allow limiting the total number of connections allocated. One global counter counts the number of current active connections; and the other keeps track of the number of unallocated connection objects. The second counter is used to govern the total number of connection object that can be created to some arbitrary limit. When a new Table 2 is allocated, this counter is adjusted downward to account for the number of objects the newly allocated table represents. On the flip side, when Internet Mobility Protocol releases a Table 2 instance back to the memory pool, the counter is adjusted upward with the number of connection objects that are being released.

Work Flow

Work is requested by local clients through the Internet Mobility ProtocolRequestWork( ) function. Once the work is validated and placed on the global work queue, the Internet Mobility ProtocolWorkQueueEligible( ) function is invoked. If in a threaded environment, the Internet Mobility Protocol worker thread is signaled (marked eligible) and control is immediately returned to the calling entity. If in a synchronous environment, the global work queue is immediately run to process any work that was requested. Both methods end up executing the Internet Mobility ProtocolProcessWork( ) function. This is the main dispatching function for processing work.

Since only one thread at a time may be dispatching work from the global queue in the example embodiment, a global semaphore may be used to protect against reentrancy. Private Internet Mobility Protocol work can post work directly to the global work queue instead of using the Internet Mobility ProtocolRequestWork( ) function.

A special case exists for SEND type work objects. To insure that the semantics of Unreliable Datagrams is kept, each SEND type work object can be queued with an expiry time or with a retry count. Work will be aged based on the expiry time. If the specified timeout occurs, the work object is removed from the connection specific queue, and is completed with an error status. If the SEND object has already been coalesced into the data path, the protocol allows for the removal of any SEND object that has specified a retry count. Once the retry count has been exceeded, the object is removed from the list of elements that make up the specific frame, and then returned to the requester with the appropriate error status.

Connection Startup

Internet Mobility Protocol includes a very efficient mechanism to establish connections between peers. Confirmation of a connection can be determined in as little as a three-frame exchange between peers. The initiator sends an IMP SYNC frame to alert its peer that it is requesting the establishment of a connection. The acceptor will either send an IMP ESTABLISH frame to confirm acceptance of the connection, or send an IMP ABORT frame to alert the peer that its connection request has been rejected. Reason and status codes are passed in the IMP ABORT frame to aid the user in decipher the reason for the rejection. If the connection was accepted, an acknowledgement frame is sent (possibly including protocol data unit or control data) and is forwarded to the acceptor to acknowledge receipt of its establish frame.

To further minimize network traffic, the protocol allows user and control data to be included in the initial handshake mechanism used at connection startup. This ability can be used in an insecure environment or in environments where security is dealt with by a layer below, such that the Internet Mobility Protocol can be tailored to avert the performance penalties due to double security authentication and encryption processing being done over the same data path.

Data Transfer

Internet Mobility Protocol relies on signaling from the NAI to detect when a frame has been delivered to the network. It uses this metric to determine if the network link in question has been momentarily flow controlled, and will not submit the same frame for retransmission until the original request has been completed. Some network drivers however lie about the transmission of frames and indicate delivery prior to submitting them to the network. Through the use of semaphores, the Internet Mobility Protocol layer detects this behavior and only will send another datagram until the NAL returns from the original send request.

Once a frame is received by Internet Mobility Protocol, the frame is quickly validated, then placed on an appropriate connection queue. If the frame does not contain enough information for Internet Mobility Protocol to discern its ultimate destination, the frame is placed on the Internet Mobility Protocol socket queue it frame was received on, and then that socket queue is place on the global work queue for subsequence processing. This initial demultiplexing allows received work to be dispersed rapidly with limited processing overhead.

Acquiescing

To insure minimal use of network bandwidth during periods of retransmission and processing power on the Mobility Management Server **102**, the protocol allows the Mobility Management Server **102** to "acquiesce" a connection. After a user configurable period of time, the Mobility Management Server **102** will stop retransmitting frames for a particular connection if it receives no notification from the corresponding Mobile End System **104**. At this point, the Mobility Management Server **102** assumes that the Mobile End System **104** is in some unreachable state (i.e. out of range, suspended, etc), and places the connection into a dormant state. Any further work destined for this particular connection is stored for future delivery. The connection will remain in this state until one of the following conditions are met:

- Mobility Management Server **102** receives a frame from the Mobile End System **104**, thus returning the connection to its original state;
- a lifetime timeout has expired;
- an inactivity timeout has expired; or
- the connection is aborted by the system administrator.

In the case that the Mobility Management Server **102** receives a frame from the Mobile End System **104**, the connection continues from the point it was interrupted. Any work that was queued for the specific connection will be forwarded, and the state will be resynchronized. In any of the other cases, the Mobile End System **104** will be apprised of the termination of the connection once it reconnects; and work that was queued for the Mobile End System **104** will be discarded.

Connect and Send Requests

FIGS. **10A–10C** together are a flowchart of example connect and send request logic formed by Internet mobility engine **244**. In response to receipt from a command from RPC engine **240**, the Internet Mobility Protocol engine **244** determines whether the command is a "connect" request (decision block **602**). If it is, engine **244** determines whether connection resources can be allocated (decision block **603**). If it is not possible to allocate sufficient connection resources ("no" exit to decision block **603**), engine **244** declares an error (block **603***a*) and returns. Otherwise, engine **244** performs a state configuration process in preparation for handling the connect request (block **603***b*).

For connect and other requests, engine **244** queues the connect or send request and signals a global event before return to the calling application (block **604**).

To dispatch a connect or send request from the Internet Mobility Protocol global request queue, engine **244** first determines whether any work is pending (decision block **605**). If no work is pending ("no" exit to decision block **605**), engine **244** waits for the application to queue work for the connection by going to FIG. **10C**, block **625** (block **605***a*). If there is work pending ("yes" exit to decision block **605**), engine **244** determines whether the current state has been established (block **606**). If the state establish has been achieved ("yes" exit to decision block **606**), engine **244** can skip steps used to transition into establish state and jump to decision block **615** of FIG. **10B** (block **606***a*). Otherwise, engine **244** must perform a sequence of steps to enter establish state ("no" exit to decision block **606**).

In order to enter establish state, engine **244** first determines whether the address of its peer is known (decision block **607**). If not, engine **244** waits for the peer address while continuing to queue work and transitions to FIG. **10C** block **625** (block **607***a*). If the peer address is known ("yes" exit to decision block **607**), engine **244** next tests whether the requisite security context has been acquired (decision block **608**). If not, engine **244** must wait for the security context while continuing to queue work and transitioning to block **625** (block **608***a*). If security context has already been acquired ("yes" exit to decision block **608**), engine **244** declares a "state pending" state (block **608***b*), and then sends an Internet Mobility Protocol sync frame (block **609**) and starts a retransmit timer (block **610**). Engine **244** determines whether the corresponding established frame was received (block **611**). If it was not ("no" exit to decision block **611**), engine **244** tests whether the retransmit time has expired (decision block **612**). If the decision block has not expired ("no" exit to decision block **612**), engine **244** waits and may go to step **625** (block **613**). Eventually, if the established frame is never received (as tested for by block **611**) and a total retransmit time expires (decision block **614**), the connection may be aborted (block **614***a*). If the established is eventually received ("yes" exit to decision block **611**), engine **244** declares a "state established" state (block **611***a*).

Once state establish has been achieved, engine **244** tests whether the new connection has been authenticated (decision block **615**). If it has not been, engine **244** may wait and transition to step **625** (block **616**). If the connection has been authenticated ("yes" exit to decision block **615**), engine **244** tests whether authentication succeeded (decision block **617**). If it did not ("no" exit to decision block **617**), the connection is aborted (block **614***a*). Otherwise, engine **244** tests whether the peer transmit window is full (decision block **618**). If it is ("yes" exit to decision block **618**), engine **244** waits for acknowledgment and goes to step **625** (decision block **619**). If the window is not full ("no" exit to decision block **618**), engine **244** creates an Internet Mobility Protocol data frame (block **620**) and sends it (block **621**). Engine **244** then determines if the retransmit timer has started (decision block **622**). If no, engine **244** starts the retransmit timer (block **623**). Engine **244** loops through blocks **618–623** until there is no more data to send (as tested for by decision block **624**). Engine **244** then returns to a sleep mode waiting for more work and returns to the global dispatcher (block **625**).

Termination

FIG. **11** is a flowchart of example steps performed by Internet Mobility Protocol engine **244** to terminate a connection. In response to a "terminate connection" request (block **626**), the engine queues the request to its global work queue and returns to the calling application (block **626***a*). The terminate request is eventually dispatched from the

Internet Mobility Protocol process global work queue for execution (block 627) Engine 244 examines the terminate request and determines whether the terminate request should be immediate or graceful (decision block 628) If immediate ("abort" exit to decision block 628), engine 244 immediately aborts the connection (block 629) If graceful ("graceful" exit to decision block 628), engine 244 declares a "state close" state (block 628*a*), and sends an Internet Mobility Protocol "Mortis" frame (block 630) to indicate to the peer that the connection is to close Engine 244 then declares a "Mortis" state (block 630*a*) and starts the retransmit timer (block 631) Engine 244 tests whether the response of "post mortem" frame has been received from the peer (decision block 632) If not ("no" exit to decision block 632), engine 244 determines whether a retransmit timer has yet expired (decision block 633) If the retransmit timer is not expired ("no" exit to decision block 633), engine 244 waits and proceeds to step 637 (block 634) If the retransmit timer has expired ("yes" exit to decision block 633), engine 244 determines whether the total retransmit time has expired (decision block 635) If the total time is not yet expired ("no" exit to decision block 635), control returns to block 630 to resent the Mortis frame If the total retransmit time has expired ("yes" exit to decision block 635), engine 244 immediately aborts the connection (block 635*a*)

Once a "post mortem" responsive frame has been received from the peer ("yes" exit to decision block 632). engine 244 declares a "post mortem" state (block 632*a*), releases connection resources (block 636), and returns to sleep waiting for more work (block 637)

Retransmission

FIG 12 is a flowchart of example "retransmit" events logic performed by Internet Mobility Protocol engine 244 In the event that the retransmit timer has expired (block 650), engine 244 determines whether any frames are outstanding (decision block 651) If no frames are outstanding ("no" exit to decision block 651), engine 244 dismisses the timer (block 652) and returns to sleep (block 660) If, on the other hand, frames are outstanding ("yes" exit to decision block 651), engine 244 determines whether the entire retransmit period has expired (decision block 653) If it has not ("no" exit to decision block 653), the process returns to sleep for the difference in time (block 654) If the entire retransmit time period has expired ("yes" exit to decision block 653), engine 244 determines whether a total retransmit period has expired (decision block 655) If it has ("yes" exit to decision block 655) and this event has occurred in the Mobility Management Server engine 244' (as opposed to the Mobile End System engine 244), a dormant state is declared (decision block 656, block 656*a*) Under these same conditions, the Internet Mobility Protocol engine 244 executing on the Mobile End System 104 will abort the connection (block 656*b*)

If the total retransmit period is not yet expired ("no" exit to decision block 655), engine 244 reprocesses the frame to remove any expired data (block 657) and then retransmits it (block 658)—restarting the retransmit timer as it does so (block 659) The process then returns to sleep (block 660) to wait for the next event

Internet Mobility Protocol Expiration of a PDU

FIG 12 block 657 allows for the requesting upper layer interface to specify a timeout or retry count for expiration of any protocol data unit (i e a SEND work request) submitted for transmission to the associated peer By use of this functionality, Internet Mobility Protocol engine 244 maintains the semantics of unreliable data and provides other capabilities such as unreliable data removal from retransmitted frames Each PDU (protocol data unit) 506 submitted by the layer above can specify a validity timeout and/or retry count for each individual element that will eventually be coalesced by the Internet Mobility Protocol engine 244 The validity timeout and/or retry count (which can be user-specified for some applications) are used to determine which PDUs 506 should not be retransmitted but should instead be removed from a frame prior to retransmission by engine 244

The validity period associated with a PDU 506 specifies the relative time period that the respective PDU should be considered for transmission. During submission, the Internet Mobility Protocol RequestWork function checks the expiry timeout value If it is non-zero, an age timer is initialized. The requested data is then queued on the same queue as all other data being forwarded to the associated peer. If the given PDU 506 remains on the queue for longer than the time period specified by the validity period parameter, during the next event that the queue is processed, the a status code of "timeout failure" rather than being retransmitted when the frame is next retransmitted This algorithm ensures that unreliable data being queued for transmission to the peer will not grow stale and/or boundlessly consume system resources

In the example shown in FIG. 12A, three separate PDUs 506 are queued to Internet Mobility Protocol engine 244 for subsequent processing PDU 506(1) is queued without an expiry time denoting no timeout for the given request PDU 506(2) is specified with a validity period of 2 seconds and is chronologically queued after PDU 506(1) PDU 506(*n*) is queued 2 5 seconds after PDU 506(2) was queued Since the act of queuing PDU 506(*n*) is the first event causing processing of the queue and PDU 506(2) expiry time has lapsed, PDU 506(2) is removed from the work queue, completed locally and then PDU 506(*n*), is placed on the list If a validity period was specified for PDU 506(*n*) the previous sequence of events would be repeated Any event (queuing, dequeuing, etc) that manipulates the work queue will cause stale PDUs to be removed and completed

As described above, PDUs 506 are coalesced by the Internet Mobility Protocol Engine 244 transmit logic and formatted into a single data stream Each discrete work element, if not previously expired by the validity timeout, is gathered to formulate Internet Mobility Protocol data frames Internet Mobility Protocol Engine 244 ultimately sends these PDUs 506 to the peer, and then places the associated frame on a Frames-Outstanding list If the peer does not acknowledge the respective frame in a predetermined amount of time (see FIG 12 showing the retransmission algorithm), the frame is retransmitted to recover from possibly a lost or corrupted packet exchange Just prior to retransmission, the PDU list that the frame is comprised of is iterated through to determine if any requests were queued with a retry count If the retry count is non zero, and the value is decremented to zero, the PDU 506 is removed from the list, and the frames header is adjusted to denote the deletion of data In this fashion, stale data, unreliable data, or applications employing their own retransmission policy are not burdened by engine 244's retransmission algorithm.

In the FIG 12B example, again three separate PDUs 506 are queued to Internet Mobility Protocol engine 244 for subsequent processing PDU 506(1) is queued without a retry count This denotes continuous retransmission attempts or guaranteed delivery level of service PDU 506(2) is queued with a retry count of 1 and is chronologically queued after PDU 506(1) PDU 506(*n*) is queued sometime after PDU 506(2) At this point, some external event (e g , upper layer coalesce timer, etc) causes engine 244's send logic to

generate a new frame by gathering enough PDUs **506** from the work queue to generate an Internet Mobility Protocol data frame **500**. The frame header **503** is calculated and stamped with a retry ID of 0 to denote that this is the first transmission of the frame. The frame is then handed to the NAL layer for subsequent transmission to the network. At this point a retransmit timer is started since the frame in question contains a payload. For illustration purposes it is assumed that an acknowledgement is not received from the peer for a variety of possible reasons before the retransmit timer expires. The retransmit logic of engine **244** determines that the frame **500** in question is now eligible for retransmission to the network. Prior to resubmitting the frame to the NAL layer, engine **244**'s retransmit logic iterates through the associated list of PDUs **506**. Each PDU's retry count is examined and if non-zero, the count is decremented. In the process of decrementing PDU **506(2)**'s retry count, the retry count becomes zero. Because PDU **506(2)**'s retry count has gone to zero, it is removed from the list and completed locally with a status of "retry failure." The frame header **503** size is then adjusted to denote the absence of the PDU **506(2)**'s data. This process is repeated for all remaining PDUs. Once the entire frame **500** is reprocessed to produce an "edited" frame **500'**, the retry ID in the header is incremented and the resultant datagram is then handed to the NAL layer for subsequent (re)transmission.

Reception

FIGS. **13A–13D** are a flowchart of example steps performed by Internet Mobility Protocol engine **244** in response to receipt of a "receive" event. Such receive events are generated when an Internet Mobility Protocol frame has been received from network **108**. In response to this receive event, engine **244** pre-validates the event (block **670**) and tests whether it is a possible Internet Mobility Protocol frame (decision block **671**). If engine **244** determines that the received frame is not a possible frame ("no" exit to decision block **671**), it discards the frame (block **672**). Otherwise ("yes" exit to decision block **671**), engine **244** determines whether there is a connection associated with the received frame (decision block **673**). If there is a connection associated with the received frame ("yes" exit to decision block **673**), engine **244** places the work on the connection receive queue (block **674**), marks the connection as eligible to receive (block **675**), and places the connection on the global work queue (block **676**). If no connection has yet been associated with the received frame ("no" exit to decision block **673**), engine **244** places the received frame on the socket receive queue (block **677**) and places the socket receive queue on the global work queue (block **678**). In either case, engine **244** signals a global work event (block **679**). Upon dispatching of a "receive eligible" event from the global work queue (see FIG. **13B**), engine **244** de-queues the frame from the respective receive queue (block **680**). It is possible that more then one IMP frame is received and queued before the Internet Mobility Protocol engine **244** can start de-queuing the messages. Engine **244** loops until all frames have been de-queue (blocks **681**, **682**). Once a frame has been de-queued ("yes" exit to decision block **681**), engine **244** validates the received frame (block **683**) and determines whether it is okay (decision block **684**). If the received frame is invalid, engine **244** discards it (block **685**) and de-queues the next frame from the receive queue (block **680**). If the received frame is valid ("yes" exit to decision block **684**), engine **244** determines whether it is associated with an existing connection (block **686**). If it is not ("no" exit to decision block **686**), engine **244** tests whether it is a sync frame (decision block **687**). If it is not a sync frame ("no" exit to decision block **687**), the frame is discarded (block **685**). If, on the other hand, a sync frame has been received ("yes" exit to decision block **687**), engine-**244** processes it using a passive connection request discussed in association with FIGS. **14A** and **14B** (block **688**).

If the frame is associated with a connection ("yes" exit to decision block **686**), engine **244** determines whether the connection state is still active and not "post mortem" (decision block **689**). If the connection is already "post mortem," the frame is discarded (block **685**). Otherwise, engine **244** parses the frame (block **690**) and determines whether it is an abort frame (decision block **691**). If the frame is an abort frame, engine **244** immediately aborts the connection (block **691***a*). If the frame is not an abort frame ("yes" exit to decision block **691**), engine **244** processes acknowledgment information and releases any outstanding send frames (block **692**). Engine **244** then posts the frame to any security subsystem for possible decryption (block **693**). Once the frame is returned from the security subsystem engine **244** processes any control data (block **694**). Engine **244** then determines whether the frame contains application data (decision block **695**). If it does, this data is queued to the application layer (block **696**). Engine **244** also determines whether the connection's state is dormant (block **697** and **697***a*—this can happen on Mobility Management Server engine **244'** in the preferred embodiment), and returns state back to established.

If the frame is possibly a "Mortis" frame ("yes" exit to decision block **698**), engine **244** indicates a "disconnect" to the application layer (block **699**) and enters the "Mortis" state (block **699***a*). It sends a "post mortem" frame to the peer (block **700**), and enters the "post mortem" state (block **700***a*). Engine **244** then releases connection resources (block **701**) and returns to sleep waiting for more work (block **702**). If the parsed frame is a "post mortem" frame ("yes" exit to decision block **703**), blocks **700***a*, **701**, **702** are executed. Otherwise, control returns to block **680** to dequeue the next frame from the receive queue (block **704**).

Passive Connections

Blocks **14A–14B** are together a flowchart of example steps performed by Internet Mobility Protocol engine **244** in response to a "passive connection" request. Engine **244** first determines whether there is another connection for this particular device (block **720**). If there is ("yes" exit to decision block **720**), the engine determines whether it is the initial connection (decision block **721**). If peer believes the new connection is the initial connection ("yes" exit to decision block **721**), engine **244** aborts the previous connections (block **722**). If not the initial connection ("no" exit to decision block **721**), engine **244** tests whether the sequence and connection ID match (decision block **723**). If they do not match ("no" exit to decision block **723**), control returns to decision block **720**. If the sequence and connection ID do match ("yes" exit to decision block **723**), engine **244** discards duplicate frames (block **724**) and returns to step **680** of FIG. **13B** (block **725**).

If there is no other connection ("no" exit to decision block **720**), engine **244** determines whether it can allocate connection resources for the connection (decision block **726**). If it cannot, an error is declared ("no" exit to decision block **726**, block **727**), and the connection is aborted (block **728**). If it is possible to allocate connection resources ("yes" exit to decision block **726**), engine **244** declares a "configure" state (block **726***a*) and acquires the security context for the connection (block **730**). If it was not possible to acquire sufficient security context ("no" exit to decision block **731**), the connection is aborted (block **728**). Otherwise, engine

244 sends an established frame (block 732) and declares the connection to be in state "establish" (block 732*a*) Engine 244 then starts a retransmitter (block 733) and waits for the authentication process to conclude (block 734) Eventually, engine 244 tests whether the device and user have both been authenticated (block 735) If either the device or the user is not authenticated, the connection is aborted (block 736) Otherwise, engine 244 indicates the connection to the listening application (block 737) and gets the configuration (block 738) If either of these steps do not succeed, the connection is aborted (decision block 739, block 740) Otherwise, the process returns to sleep waiting for more work (block 741)

Abnormal Termination

FIGS 15A and 15B are a flowchart of example steps performed by the Internet Mobility Protocol engine 244 in response to an "abort" connection request Upon receipt of such a request from another process (block 999) and are dispatched via the queue (block 1000), engine 244 determines whether the connection is associated with the request (decision block 1001) If it is ("yes" exit to decision block 1001), engine 244 saves the original state (block 1002) and declares an "abort" state (block 1002*a*) Engine 244 then determines whether the connection was indicated to any listening application (decision block 1003)—and if so, indicates a disconnect to that listening application (block 1004) Engine 244 then declares a "post mortem" state (block 1003*a*), releases the resources previously allocated to the particular connection (block 1005), and tests whether the original state is greater than the state pending (decision block 1006) If not ("no" exit to decision block 1006), the process transitions to block 1002 to return to the calling routine (block 1007) Otherwise, engine 244 determines whether the request is associated with a received frame (decision block 1008) If the abort request is associated with a received frame, and the received frame is an abort frame (decision block 1009), the received frame is discarded (block 1010) Otherwise engine 244 will send an abort frame (block 1011) before returning to the calling routine (block 1012).

Roaming Control

Referring once again to FIG. 1, mobile network 108 may comprise a number of different segments providing different network interconnects (107*a*–107*k* corresponding to different wireless transceivers 106*a*–106*k*) In accordance with another aspect of a presently preferred exemplary embodiment of the present invention, network 108 including Mobility Management Server 102 is able to gracefully handle a "roaming" condition in which a Mobile End System 104 has moved from one network interconnect to another Commonly, network 108 topographies are divided into segments (subnets) for management and other purposes These different segments typically assign different network (transport) addresses to the various Mobile End Systems 104 within the given segment

It is common to use a Dynamic Host Configuration Protocol (DHCP) to automatically configure network devices that are newly activated on such a subnet. For example, a DHCP server on the sub-net typically provides its clients with (among other things) a valid network address to "lease" DHCP clients may not have permanently assigned, "hard coded" network addresses Instead, at boot time, the DHCP client requests a network address from the DHCP server The DHCP server has a pool of network addresses that are available for assignment. When a DHCP client requests an network address, the DHCP server assigns, or leases, an available address from that pool to the client The assigned network address is then "owned" by the client for a specified period ("lease duration") When the lease expires, the network address is returned to the pool and becomes available for reassignment to another client In addition to automatically assigning network addresses, DHCP also provides netmasks and other configuration information to clients running DHCP client software More information concerning the standard DHCP protocol can be found in RFC2131

Thus, when a Mobile End System 104 using DHCP roams from one subnet to another, it will appear with a new network address In accordance with a presently preferred exemplary embodiment of the present invention, Mobile End Systems 104 and Mobility Management Server 102 take advantage of the automatic configuration functionality of DHCP, and coordinate together to ensure that the Mobility Management Server recognizes the Mobile End System's "new" network address and associates it with the previously established connection the Mobility Management Server is proxying on its behalf

The preferred embodiment uses standard DHCP Discover/Offer client-server broadcast messaging sequences as an echo request-response, along with other standard methodologies in order to determine if a Mobile End System 104 has roamed to a new subnet or is out of range In accordance with the standard DHCP protocol, a Mobile End System 104 requiring a network address will periodically broadcast client identifier and hardware address as part of a DHCP Discover message The DHCP server will broadcast its Offer response (this message is broadcast rather than transmitted specifically to the requesting Mobile End System because the Mobile End System doesn't yet have a network address to send to) Thus, any Mobile End System 104 on the particular subnet will pick up any DHCP Offer server response to any other Mobile End System broadcast on the same subnet

A presently preferred exemplary embodiment of present invention provides DHCP listeners to monitor the DHCP broadcast messages and thereby ascertain whether a particular Mobile End System 104 has roamed from one subnet to another and is being offered the ability to acquire a new network address by DHCP. FIG 16 shows example DHCP listener data structures For example, a Mobile End System listener data structure 902 may comprise:

- a linked list of server data structures,
- an integer transaction ID number (xid),
- a counter ("ping"), and
- a timeout value

A server data structure 904 may comprise a linked list of data blocks each defining a different DHCP server, each data block comprising:

- a pointer to next server,
- a server ID (network address of a DHCP server),
- an address (giaddr) of a BOOTP relay agent recently associated with this DHCP server,
- a "ping" value (socket->ping), and
- a flag

These data-structures are continually updated based on DHCP broadcast traffic appearing on network 108 The following example functions can be used to maintain these data structures:

- roamCreate( ) [initialize variables]
- roamDeinitialize( ) [delete all listeners]
- roamStartIndications( ) [call a supplied callback routine when a Mobile End System has roamed or changed interfaces, to give a registrant roaming indications]

roamStopIndications( ) [remove the appropriate callback from the list, to stop giving a registrant roaming indications]

Interface Change [callback notification from operating system indicating an interface has changed its network address]

Listener Signal [per-interface callback from a Listener indicating a roaming or out-of-range or back-in-range condition]

Additionally, a refresh process may be used to update Listeners after interface changes

In the preferred embodiment, all Mobile End Systems **104** transmit the same Client Identifier and Hardware Address in DHCP Discover requests. This allows the listener data structures and associated processes to distinguish Mobile End System-originated Discover requests from Discover requests initiated by other network devices. Likewise, the DHCP server will broadcast its response, so any Mobile End System **104** and/or the Mobility Management Server **102** will be able to pick up the DHCP server Offer response to any other Mobile End System. Since multiple DHCP servers can respond to a single DHCP Discover message, the listener data structures shown in FIG. **16** store each server response in a separate data block, tied to the main handle via linked list.

Upon receiving a Discover request having the predetermined Client Hardware Address and Client Identifier, the preferred embodiment recognizes this request as coming from a Mobile End System **104**. If the message also has a BOOTP relay address set to zero, this indicates that the message originated on the same subnet as the listener. Listeners may ignore all DHCP Offers unless they have a transaction ID (xid) matching that of a Discover message recently sent by a Mobile End System **104**. The listener can determine that a Mobile End System **104** has roamed if any response comes from a known server with a new BOOTP relay agent ID and/or offered network address masked with an offered subnet mask. Listeners add new servers to the FIG. **16** data structures only after receiving a positive response from an old server. If a listener receives responses from new server(s) but none from an old server, this may indicate roaming (this can be a configurable option). If the listener fails to receive responses from new or old servers, the listener is out of range (this determination can be used to signal an upper layer such as an application to halt or reduce sending of data to avoid buffer overflow).

If the listener never receives a response from any server, there is no point of reference and thus impossible to determine whether roaming has occurred. This condition can be handled by signaling an error after a timeout and allowing the caller to retry the process. The preferred embodiment determines that a Mobile End System **104** has roamed if any response has come from a known server with a new BOOTP relay agent ID (or a new offered network address when masked with offered subnet mask). If the listener data structures see responses from new servers but none from an old server, it is possible that roaming has occurred, but there must be a delay before signaling, in order to wait for any potential responses from the old servers. If there are no responses from new or old servers, then the Mobile End System **104** is probably out of range and Mobility Management Server **102** waits for it to come back into range.

FIG. **17** is a flowchart of example steps of a Listener process of the preferred embodiment. Referring to FIG. **17**, a DHCP listener process is created by allocating appropriate memory for the handle, opening NAI sockets for the DHCP client and server UDP ports, and setting receive callbacks for both. A timer is then set (block **802**) and then the process enters the "Wait" state to wait for a roaming related event (block **804**). Three external inputs can trigger an event:

a DHCP server packet is received;

a DHCP client packet sent by another Mobile End System is received

a timer timeout occurs

If a DHCP server packet has been received, the packet is examined to determine whether its client identifier matches the predetermined client ID (decision block **806**). If it does not, it is discarded. However, if the packet does contain the predetermined ID, a test is performed to determine whether the packet is a DHCP Offer packet (decision block **808**). Offer packets are rejected unless they contain a transaction ID matching a recently sent DHCP Discover sequence.

If the packet transaction ID matches (block **810**), then a test is made as to whether the server sending the DHCP offer packet is known (i.e., the server ID is in the listener data structure shown in FIG. **16**) (block **812**). If the server ID is not on the list ("no" exit to decision block **812**), it is added to the list and marked as "new" (or "first" if it is the first server on the list) (block **822**). If the server is already on the list ("Y" exit to decision block **812**), a further test is performed to determine whether the packet BOOTP relay address ("GIADDR") matches the server address ("GIADDR") (decision block **814**). If there is no match, then the Offer packet must be originating from a different subnet, and it is determined that a "hard roam" has occurred (block **816**). The caller application is signaled that there has been a roam. If, on the other hand, decision block **814** determines there is a match in BOOTP relay addresses, then no roam has occurred, the listener process stamps the server receive time, resets "new" flags for all other servers on the list, and stores the current ping number with the server (block **818**, **820**). The process then returns to "wait" period.

If the event is a received client packet, the listener process determines whether the packet has the predetermined client ID, is a DHCP Discover packet and has a BOOTP relay address (GIADDR) of **0** (blocks **824**, **826**, **828**). These steps determine whether the received packet is DHCP Discover message sent by another Mobile End System **104** on the same sub-net as the listener. If so, the listener process then sets the transaction ID to the peer's transaction ID (block **830**) for use in comparing with later-received DHCP Offer packets, calls a ping check (block **834**) and resets the timer (block **836**).

In response to a timer timeout, the process calls a "ping check" (block **838**). "Pings" in the preferred embodiment are DHCP Discover packets with a random new xid. Example steps for this ping check **838** are shown in FIG. **17A**. The purpose of the ping check routine is to determine if a "soft roam" condition has occurred (i.e., a Mobile End System has temporarily lost and then regained contact with a sub-net, but has not roamed to a different sub-net). The process determines whether there is a sub-net roam condition, an out-of-range condition, or a "no server" condition. In other words:

Has a Mobile End System roamed from one sub-net to another?

Is a Mobile End System out of range?

Is a DHCP server absent?

These conditions are determined by comparing Mobile End System prior "ping" response with the current "ping" response (decision blocks **846**, **850**). For example, if the current ping number minus the old server's last ping response is greater than the sub-net server pings and there is

at least one server marked "new," there has been a sub-net roam to a different server The result of this logic is to either signal a subset roam, and out of range condition or a no server condition (or none of these) to the calling process

FIG. 18 shows a flowchart of example steps performed by a Mobile End System 104 roaming control center To enable roaming at the Mobile End System 104, the list of known addresses is initialized to zero (block 850) and an operating system interface change notification is enabled (block 852) The process then calls the operating system to get a list of current addresses that use DHCP (block 854) All known addresses no longer in the current list have their corresponding listeners closed (block 856) Similarly, the process opens listeners on all current but not known interfaces (block 858) The process then signals "roam" to registrants (block 860)

When the listener process of FIG 17 signals (block 862), the process determines whether the signal indicates a "roam", "out of range" or "back in range" condition (decision block 864, 870, 874) A roam signal ("yes" exit to decision block 864) causes the process to close corresponding listener 866 and call the operating system to release and renew DHCP lease to a network address (block 868). If the listener signals "out of range" (decision block 870), the process signals this condition to registrants (block 872). If the signal is a "back in range" (decision block 874), then this condition is signaled to all registrants (block 876) Upon receiving a disabled roam command (block 878), the process closes all listeners (block 880) and disables the operating system interface change notification (block 882)

EXAMPLES

A presently preferred exemplary embodiment of present invention finds application in a variety of real-world situations For example:

Intermittently Connected Portable Computer

Many businesses have employees who occasionally telecommute or work from home Such employees often use laptop computers to get their work done While at work, the employees typically connect their laptop computers to a local area network such as an Ethernet through use of a docking port or other connector The LAN connection provides access to network services (e g , printers, network drives) and network applications (e g , database access, email services)

Now suppose an employee working on a project needs to go home for the evening and wants to resume working from home The employee can "suspend" the operating system and applications running on the laptop computer, pack up the laptop computer, and bring the laptop computer home

Once home, the employee can "resume" the operating system and applications running on the laptop computer, and reconnect to the office LAN via a dialup connection and/or over the Internet The Mobility Management Server (which continued to proxy the laptop computer vis-a-vis the network and its applications during the time the laptop computer was temporarily suspended) can re-authenticate the laptop computer and resume communicating with the laptop computer.

From the perspective of the employee now working from home, all of the network drive mappings, print services, email sessions, database queries, and other network services and applications, are exactly where the employee left them at the office Furthermore, because the Mobility Management Service continued to proxy the laptop computer's sessions, none of those network applications terminated the laptop computer's sessions during the time the employee was traveling from the office to home The exemplary embodiment of the invention thus provides efficient persistence of session across the same or multiple network mediums that is very powerful and useful in this and other contexts.

Mobile Inventory and Warehouse Application

Imagine a large warehouse or retail chain Within this campus, inventory workers use vehicle mounted (i e , trucks and forklifts) personal laptop computers and handheld data collection units and terminals to perform inventory management of goods Warehouse and retail workers are often inexperienced computer users that do not understand network sub-nets and require management supervision A presently preferred exemplary embodiment of present invention allows the creation of a turnkey system that hides the complexity of the mobile network from the warehouse users. The users can move in and out of range of access points, suspend and resume their Mobile End Systems 104, and change locations without concern for host sessions, network addresses, or transport connections In addition, the management software on the Mobility Management Server 102 provides management personnel with metrics such as number of transactions, which may be used to gauge worker productivity Management can also use the network sub-net and access points to determine worker's last known physical location.

Mobile Medical Application

Imagine a large hospital using radio LAN technology for network communications between several buildings Each building is on a unique sub-net. A presently preferred exemplary embodiment of present invention enables nurses and doctors to move from room to room with handheld personal computers or terminals—reading and writing patient information in hospital databases Access to the most recent articles on medication and medical procedures is readily available through the local database and the World Wide Web While in the hospital, pagers (one and two way) are no longer required since a presently preferred exemplary embodiment of the present invention allows continuous connection to the Mobile End System 104 Messages can be sent directly to medical personnel via the Mobile End System 104 As in the case with warehouse workers, medical personnel are not required to understand the mobile network they are using. In addition, the Mobile End System 104 allows medical personnel to disable radio transmission in area where radio emissions are deemed undesirable (e g , where they might interfere with other medical equipment)—and easily resume and reconnect where they left off

Trucking and Freight

Freight companies can a presently preferred exemplary embodiment of use the present invention to track inventory. While docked at a warehouse, the Mobile End System 104 may use LAN technology to update warehouse inventories While away from local services, the Mobile End System 104 can use Wide Area WAN services such as CDPD and ARDIS to maintain real time status and location of inventory The Mobile End System 104 automatically switches between network infrastructures—hiding the complexity of network topology from vehicle personnel

Mobile Enterprise

Corporate employees may use the system in accordance with a presently preferred exemplary embodiment of present invention for access to E-mail, web content and messaging services while within an enterprise campus that has invested in an infrastructure such as 802 11 The cost of ownership is reduced since pager service and other mobile device services are no longer required The purchase of mobile infrastructure is a one time capital expense as opposed to the costly

"pay-per-use" model offered by many existing mobile device services

IP Multiplication

If an organization has a LAN that needs to be connected to the Internet, the administrator of the LAN has two choices: get enough globally assigned addresses for all computers on the LAN, or get just a few globally assigned addresses and use the Mobility Management Server 102 in accordance with a presently preferred exemplary embodiment of the present invention as an address multiplier Getting a large number of IP addresses tends to be either expensive or impossible A small company using an Internet Service Provider (ISP) for access to the Internet can only use the IP addresses the ISP assigns—and the number of IP addresses limits the number of computers that can be on the Internet at the same time. An ISP also charges per connection, so the more computers that need to be on the Internet, the more expensive this solution becomes

Using the Mobility Management Server 102 in accordance with the present invention as an address multiplier could solve many of these problems The enterprise could put the Mobility Management Server 102 on hardware that is connected to the Internet via an ISP Mobile End Systems 104 could then easily connect Because all connection to the Internet would go through the Mobility Management Server 102, only one address from the ISP is required Thus, using a presently preferred exemplary embodiment of the present invention as an address multiplier allows the enterprise to get just a few (in many cases one) addresses and accounts from the ISP, and allows the entire LAN to have simultaneous connections to the Internet (assuming enough bandwidth is provided)

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the scope of the appended claims

What is claimed is:

1 A method for enabling secure data communication with a computing device that roams among plural data communication networks or subnetworks of the type that carry Internet Protocol (IP) data. said method comprising:

establishing access to at least one data communications network or subnetwork carrying IP data;

using said network or subnetwork to institute secure data communications between said computing device and another computing device;

upon said computing device roaming while operational to a further network or subnetwork carrying IP data, establishing access at the computing device with said further network or subnetwork to enable said computing device to communicate over said further network or subnetwork;

participating in a DHCP process with said further network or subnetwork;

learning, at least in part in response to said DHCP process, that said computing device is enabled to communicate over said further network or subnetwork;

based at least in part on said learning, informing said another computing device that said computing device has roamed and has access to said further network or subnetwork; and

using said access to said further network or subnetwork to continue said secure data communications between the computing device and said another computing device

2 The method of claim 1 wherein said computing device comprises a mobile computing device that communicates wirelessly with said at least one data communications network or subnetwork and said further network or subnetwork

3. The method of claim 1 wherein said another computing device comprises an intermediary computing device

4. The method of claim 1 wherein said another computing device comprises a server.

5. The method of claim 1 wherein said another computing device comprises a proxy server

6 The method of claim 1 wherein the at least one data communications network or subnet carrying IP data and the further data communications network or subnet carrying IP data provide a homogeneous collection of link layer data communication networks or subnetworks

7. The method of claim 1 wherein the at least one data communications network or subnet carrying IP data and the further data communications network or subnet carrying IP data provide a heterogeneous collection of link layer data communication networks or subnetworks

8 The method of claim 1 wherein said participating includes sending a discover message.

9 The method of claim 1 wherein said learning compares a network address dynamically provided by DHCP with a previously-used network address, and determines whether said addresses are different.

10 The method of claim 1 wherein said computing device performs said participation, said DHCP process dynamically provides a network address to said computing device, and said method further includes using said dynamically-provided network address to continue said secure communications via said further network or subnetwork

11 The method of claim 1 wherein said learning includes detecting, based on a network address that is dynamically provided by said DHCP process, that said computing device's network point of attachment has changed.

12 The method of claim 1 further including authenticating or reauthenticating said continued secure communications in response to learning said computing device has roamed.

13 The method of claim 1 wherein said data communications network or subnetwork is wireless, and said further data communications network or subnetwork is non-wireless

14 The method of claim 1 wherein said data communications network or subnetwork is non-wireless, and said further data communications network or subnetwork is wireless.

15 The method of claim 1 wherein said computing device supports at least one application layer session, and said roaming allows said application layer session to continue without terminating

16 The method of claim 1 further including changing routing information for said secure communications to continue

17 The method of claim 1 further including intercepting calls at the transport driver interface

18 The method of claim 1 further including intercepting calls at the socket API level

19 The method of claim 1 further including establishing said secure communications with said computing device using UDP.

20 The method of claim 1 further including issuing a ping check to determine if the computing device temporarily lost and then regained contact with a sub-network, but has not roamed to a different sub-network

21 The method of claim 1 wherein said computing device comprises a laptop computer with a display, a keyboard and a wireless interface

22 The method of claim 1 wherein said computing device comprises a mobile medical computing device

23 The method of claim 1 further including providing configurable session priorities for said computing device and/or users thereof

24 The method of claim 1 wherein the point of presence address of said computing device changes during roaming, and wherein said method further includes maintaining a constant virtual address associated with said computing device at least during roaming

25 The method of claim 1 wherein said computing device moves from one type of connection to another during roaming.

26 The method of claim 1 wherein said computing device connects to said network or subnetwork and to said further network or subnetwork using different network interfaces

27 A method, performed by a computing device while the computing device is operational, for enabling secure data communication over at least one data communication network or subnetwork of the type that carries Internet Protocol (IP) data, said method comprising:

establishing access to at least one data communications network or subnetwork carrying IP data;

using said network or subnetwork access to establish secure data communications with another computing device;

disassociating access from said network or subnetwork;

subsequent to said disassociating step, reestablishing access to said network or subnetwork;

participating in a DHCP process with said network or subnetwork;

learning, at least in part in response to said DHCP process, that said computing device has reestablished access to said network or subnetwork;

based at least in part on said learning, informing said another computing device of said reestablished access; and

using said reestablished access to continue said secure data communications with said another computing device

28 The method of claim 27 wherein said computing device comprises a mobile computing device that communicates wirelessly with said at least one data communications network or subnetwork carrying IP data

29 The method of claim 27 wherein said another computing device comprises an intermediary computing device

30 The method of claim 27 wherein said another computing device comprises a server

31 The method of claim 27 wherein said another computing device comprises a proxy server

32 The method of claim 27 wherein the at least one data communications network or subnet carrying IP data comprises a homogeneous collection of link layer data communication networks or subnetworks.

33 The method of claim 27 wherein the at least one data communications network or subnet carrying IP data comprises a heterogeneous collection of link layer data communication networks or subnetworks

34 The method of claim 27 wherein said participating includes sending a discover message

35 The method of claim 27 wherein said learning compares a network address dynamically provided by DHCP with a previously-used network address, and determines whether said addresses are different

36 The method of claim 27 wherein said computing device performs said participation, said DHCP process dynamically supplies a network address to said computing device, and said method further includes using said dynamically supplied network address to continue said secure data communications.

37 The method of claim 27 wherein said learning includes detecting, based on a network address that is dynamically provided by said DHCP process, that a network point of attachment has not changed

38 The method of claim 27 further including authenticating or reauthenticating said continued secure data communications

39 The method of claim 27 wherein said data communications network or subnetwork is wireless.

40 The method of claim 27 wherein said data communications network or subnetwork is non-wireless

41 The method of claim 27 wherein said computing device supports at least one application layer session, and said disassociating does not cause said application layer session to terminate.

42 The method of claim 27 further including intercepting calls at the transport driver interface

43 The method of claim 27 further including intercepting calls at the socket API level

44 The method of claim 27 further including establishing said secure data communications using UDP.

45 The method of claim 27 further including issuing a ping check to determine if the computing device temporarily lost and then regained contact with a sub-network, but has not roamed to a different sub-network

46 The method of claim 27 wherein said computing device comprises a laptop computer with a display, a keyboard and a wireless interface

47 The method of claim 27 wherein said computing device comprises a mobile medical computing device.

48 The method of claim 27 further including providing configurable session priorities for said computing device and/or users thereof.

49 The method of claim 27 wherein the point of presence address of said computing device changes during roaming, and wherein said method further including maintaining a virtual address associated with said computing device constant at least during roaming

50 The method of claim 27 wherein said computing device moves from one type of connection to another on said network or subnetwork

51 The method of claim 27 wherein said computing device can connect to said network or subnetwork using plural different network interfaces

52 A method for allowing a mobile computing device to move between plural IP-based networks, comprising:

connecting the mobile computing device to at least one network;

using a dynamically-supplied IP address to enable secure data communications providing application and/or transport layer sessions between said mobile computing device and another computing device;

then migrating the mobile computing device to a further network while said mobile computing device is operational;

determining, based at least in part on a DHCP process performed on said further network, that said mobile computing device may have moved; and

in response to said determining, taking further action to continue said secure data communications including said application and/or transport layer sessions between said moved mobile computing device and said another

computing device over said further network without terminating said application and/or transport layer sessions

53 A method for allowing a mobile computing device to roam across IP-based networks, comprising:

connecting the mobile computing device to at least one network medium used for data communications;

using a network layer address to facilitate secure IP-based data communications including application and/or transport layer sessions between said mobile computing device and another computing device at least in part via said network medium;

then connecting the mobile computing device to a further network medium without first rebooting said mobile computing device;

performing a DHCP process over said further network medium;

discovering, based at least in part on said DHCP process, that said mobile computing device has roamed; and

based at least in part on said discovering, using a further network layer address to continue said secure IP-based data communications between said mobile computing device and said another computing device without terminating application and transport layer sessions therebetween

54 A system for enabling secure data communication comprising:

plural data communication networks or subnetworks of the type that carry Internet Protocol (IP) data;

a mobile computing device that, while operational, roams among said plural data communication networks or subnetworks; and

another computing device that communicates with said mobile computing device via said plural data communications networks or subnetworks,

wherein said mobile computing device comprises:

at least one wireless data communications device for establishing access to said at least one data communications network or subnetwork carrying IP data and for using said network or subnetwork to institute secure wireless data communications with said another computing device, and upon said mobile computing device wirelessly roaming while operational to a further network or subnetwork carrying IP data, for establishing access with said further network or subnetwork to enable said mobile computing device to communicate over said further network or subnetwork,

a listener that participates in a DHCP process with said further network or subnetwork;

a detector coupled to said listener, said detector learning, at least in part in response to said DHCP process, that said mobile computing device is enabled to communicate over said further network or subnetwork and, based at least in part on said learning, using said access to said further network or subnetwork to continue said secure data communications between the mobile computing device and said another computing device

55 A computing device comprising:

data communicators that establishes access to at least one data communications network or subnetwork carrying Internet Protocol (IP) data over at least one data communication network or subnetwork, and which uses said access to said network or subnetwork to establish secure data communications with another computing device;

a DHCP listener that participates in a DHCP process with said network or subnetwork upon said computing device disassociating access from said network or subnetwork and then reestablishing access to said network or subnetwork, said DHCP listener learning, at least in part in response to said DHCP process, that said computing device has reestablished access to said network or subnetwork;

wherein said data communicator informs said another computing device, based at least in part on said learning, of said reestablished access and uses said reestablished access to continue said secure data communications with said another computing device

* * * * *